МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
(ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ)

Кафедра информатики и вычислительной математики

СЕТОЧНО-ХАРАКТЕРИСТИЧЕСКИЕ МЕТОДЫ ДЛЯ ЧИСЛЕННОГО РЕШЕНИЯ ЛИНЕЙНЫХ ОДНОМЕРНЫХ ГИПЕРБОЛИЧЕСКИХ УРАВНЕНИЙ И СИСТЕМ

Учебно-методическое пособие

Составитель *В.И. Голубев*

МОСКВА
МФТИ
2018

УДК 519.63

Рецензент
Кандидат физико-математических наук *А.В. Васюков*

Сеточно-характеристические методы для численного решения линейных одномерных гиперболических уравнений и систем: учебно-методическое пособие / сост. В.И. Голубев. – М.: МФТИ, 2018. – 41 с.

Работа посвящена построению численных сеточно-характеристических методов для решения линейного одномерного уравнения переноса, которое возникает в задачах переноса взвеси частиц в жидкости, газовой динамике и динамических задачах упругого тела. На примере систем уравнений акустики и линейной упругости рассмотрен подход, позволяющий проводить их решение путём последовательного решения нескольких уравнений переноса. Предназначается для студентов, изучающих информатику и вычислительную математику, в виде практики по реализации вычислительных алгоритмов на языке Си.

© Голубев В.И., 2018
© Федеральное государственное автономное образовательное учреждение высшего образования «Московский физико-технический институт (государственный университет)», 2018

Содержание

Введение

Гиперболические системы уравнений составляют основу математических моделей множества физических процессов [1, 2, 3, 4]. К ним относятся: задачи газовой динамики, линейно-упругого тела, примесного массопереноса. К сожалению, превалирующее число прикладных задач, определяемых как расчётной областью, так и начальными и граничными условиями, не имеют аналитического решения. Таким образом, важной задачей является изучение существующих и разработка новых численных методов для решения гиперболических систем уравнений.

Одним из подходов к построению численных методов решения гиперболических систем уравнений является метод расщепления по пространственным переменным, который совместно с соответствующей заменой переменных позволяет свести задачу к последовательному решению набора линейных одномерных уравнений переноса. Данная работа посвящена проблеме построения сеточных численных методов для его решения. Рассматривается общий подход, основанный на сеточно-характеристическом методе и интерполяции искомой функции полиномами на расчётном шаблоне. Описаны два различных способа повышения порядка аппроксимации схемы: расширение пространственного шаблона [5] и дополнение исходного уравнения его дифференциальным следствием [6]. Рассмотрен пример реализации численных схем на языке Си, предложены задачи для самостоятельной работы. Также показана возможность расширения компактных сеточно-характеристических схем на случай акустической системы уравнений в одномерной постановке. Показан способ перехода в инварианты Римана для системы уравнений упругости в одномерной постановке.

Работа выполнена при поддержке стипендии Президента РФ молодым учёным и аспирантам СП-1591.2016.5.

1. Линейное уравнение переноса

Рассмотрим одномерное линейное уравнение переноса:

$$u_t + \lambda u_x = 0, \tag{1}$$

где $u(x,t)$ – искомая функция, например, концентрация взве-

шенных частиц в жидкости, λ – скорость переноса. Введём замену переменных

$$\xi = x - \lambda t \tag{2}$$

и будем искать решение уравнения (1) в виде $u(\xi)$. Прямой подстановкой можно убедиться, что (1) превращается в тождество. Таким образом, если рассмотреть два последовательных момента времени t^n и $t^n + \tau$, можно записать следующее равенство:

$$u(x_m, t_n + \tau) = u(x_m - \lambda\tau, t_n). \tag{3}$$

На базе данного свойства построен класс высокоточных сеточно-характеристических численных методов [7]. Все дальнейшие выкладки будем проводить в предположении $\lambda \geq 0$, поскольку обобщение на случай отрицательного коэффициента путём замены λ на $-\lambda$ и изменения шаблона на противоположный не представляет труда.

1.1. Схемы на расширенном шаблоне

1.1.1. Построение и анализ

Согласно уравнению (3) необходимо иметь возможность восстанавливать значение искомой функции $u(x, t)$ в произвольной точке расчётной области, тогда как при численном расчёте её значения хранятся в фиксированных узлах расчётной сетки. Для решения данной задачи воспользуемся процедурой интерполяции функции полиномом заданной степени. Отметим, что все рассматриваемые схемы будут относиться к явному типу, т.е. на слое $n+1$ будет присутствовать только одна точка шаблона x_m^{n+1}.

Рассмотрим двухточечный по пространству шаблон (рис. 1), на котором можно построить полином первой степени по значениям функции в узлах x_m^n и x_{m-1}^n.

В общем виде полином первого порядка представим в виде

$$F_1(x) = a_1 x + b_1. \tag{4}$$

Коэффициенты a_1 и b_1 могут быть найдены из условия прохождения полинома $F_1(x)$ через значения функции, хранимые в узлах шаблона:

$$a_1 = \frac{u_m^n - u_{m-1}^n}{h},$$
$$b_1 = u_m^n.$$

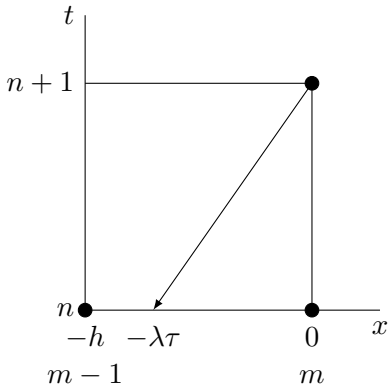

Рис. 1. Двухточечный по пространству шаблон, используемый для построения схемы первого порядка

Подставляя их в выражение (4) и используя соотношение (3), получим

$$u_m^{n+1} = u_m^n - \frac{\lambda\tau}{h}(u_m^n - u_{m-1}^n). \tag{5}$$

Наиболее простым и интуитивно ясным способом повышения порядка аппроксимации схемы по пространству является расширение сеточного шаблона.

Расширим шаблон в положительном направлении, добавив точку x_{m+1}^n. На нём может быть построен полином второй степени, в общем виде представимый соотношением

$$F_2(x) = a_2x^2 + b_2x + c_2. \tag{6}$$

Коэффициенты a_2, b_2 и c_2 могут быть найдены из условия прохождения полинома $F_2(x)$ через значения функции, хранимые в узлах шаблона:

$$a_2 = \frac{u_{m+1}^n - 2u_m^n + u_{m-1}^n}{2h^2},$$
$$b_2 = \frac{u_{m+1}^n - u_{m-1}^n}{2h},$$
$$c_2 = u_m^n.$$

Подставляя их в выражение (6) и используя соотношение (3), получим

$$u_m^{n+1} = u_m^n - \frac{\lambda\tau}{2h}(u_{m+1}^n - u_{m-1}^n) + \frac{\lambda^2\tau^2}{2h^2}(u_{m+1}^n - 2u_m^n + u_{m-1}^n). \quad (7)$$

Для дальнейшего повышения порядка схемы расширим шаблон, добавив в него точку x_{m-2}. На нём может быть построен полином третьей степени, в общем виде представимый соотношением

$$F_3(x) = a_3x^3 + b_3x^2 + c_3x + d_3. \quad (8)$$

Коэффициенты a_3, b_3, c_3 и d_3 могут быть найдены из условия прохождения полинома $F_3(x)$ через значения функции, хранимые в узлах шаблона:

$$a_3 = \frac{u_{m+1}^n + 3u_{m-1}^n - 3u_m^n - u_{m-2}^n}{6h^3},$$
$$b_3 = \frac{u_{m+1}^n - 2u_m^n + u_{m-1}^n}{2h^2},$$
$$c_3 = \frac{2u_{m+1}^n + 3u_m^n - 6u_{m-1}^n + u_{m-2}^n}{6h},$$
$$d_3 = u_m^n.$$

Подставляя их в выражение (8) и используя соотношение (3), получим

$$u_m^{n+1} = u_m^n - \frac{\lambda\tau}{6h}(2u_{m+1}^n + 3u_m^n - 6u_{m-1}^n + u_{m-2}^n) + \\ + \frac{\lambda^2\tau^2}{2h^2}(u_{m+1}^n - 2u_m^n + u_{m-1}^n) - \\ - \frac{\lambda^3\tau^3}{6h^3}(u_{m+1}^n + 3u_{m-1}^n - 3u_m^n - u_{m-2}^n). \quad (9)$$

Для дальнейшего повышения порядка схемы расширим шаблон, добавив в него точку x_{m+2}. На нём может быть построен полином четвёртой степени, в общем виде представимый соотношением

$$F_4(x) = a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4. \quad (10)$$

Коэффициенты a_4, b_4, c_4, d_4 и e_4 могут быть найдены из условия прохождения полинома $F_4(x)$ через значения функции, хранимые в узлах шаблона:

$$a_4 = \frac{6u_m^n - 4u_{m+1}^n - 4u_{m-1}^n + u_{m+2}^n + u_{m-2}^n}{24h^4},$$
$$b_4 = \frac{2u_{m-1}^n - 2u_{m+1}^n + u_{m+2}^n - u_{m-2}^n}{12h^3},$$
$$c_4 = \frac{16u_{m+1}^n - 30u_m^n + 16u_{m-1}^n - u_{m+2}^n - u_{m-2}^n}{24h^2},$$
$$d_4 = \frac{8u_{m+1}^n - 8u_{m-1}^n - u_{m+2}^n + u_{m-2}^n}{12h},$$
$$e_4 = u_m^n.$$

Подставляя их в выражение (10) и используя соотношение (3), получим

$$u_m^{n+1} = u_m^n - \frac{\lambda\tau}{12h}(8u_{m+1}^n - 8u_{m-1}^n - u_{m+2}^n + u_{m-2}^n) +$$
$$+\frac{\lambda^2\tau^2}{24h^2}(16u_{m+1}^n - 30u_m^n + 16u_{m-1}^n - u_{m+2}^n - u_{m-2}^n) -$$
$$-\frac{\lambda^3\tau^3}{12h^3}(2u_{m-1}^n - 2u_{m+1}^n + u_{m+2}^n - u_{m-2}^n) +$$
$$+\frac{\lambda^4\tau^4}{24h^4}(6u_m^n - 4u_{m+1}^n - 4u_{m-1}^n + u_{m+2}^n + u_{m-2}^n). \qquad (11)$$

▲ Исследовательские задания

1. Постройте схемы повышенных порядков аппроксимации (начиная с 5-го) путём расширения сеточного шаблона точками $x_{m-3}^n, x_{m+3}^n, x_{m-4}^n, x_{m+4}^n$, и т.д. соответственно. Отметим, что некоторые из них могут не обладать свойством устойчивости [8]. Подумайте, какие возникнут проблемы при их компьютерной реализации.

2. Как было сказано ранее, для случая $\lambda < 0$ схема первого порядка может быть получена простой заменой знака у собственного числа и перехода от x_{m-1}^n к x_{m+1}^n. Подумайте, как реализовать численную схему, в которой собственные числа как положительные, так и отрицательные. Простейший

пример – движение взвеси частиц в жидкости со скоростью потока в заданном (положительном или отрицательном) направлении.

3. Покажите, что сеточно-характеристическая схема при аппроксимации полиномом второго порядка совпадает со схемой Лакса–Вендроффа [9].

4. Покажите, что сеточно-характеристическая схема при аппроксимации полиномом третьего порядка совпадает со схемой Русанова [5].

1.1.2. Реализация на языке Си

Описанные вычислительные алгоритмы могут быть запрограммированы на любом языке программирования. Рассмотрим реализацию численных схем на расширенном шаблоне для решения одномерного линейного уравнения переноса на языке Си.

Отдельные этапы алгоритма логично вынести в функции. При таком подходе, например, можно будет обеспечить переход от одного типа интерполяции к другому изменением лишь функции интерполяции без излишнего дублирования кода. Рассмотрим файл с прототипами функций **advection.h**.

./src/advection.h

```
/* Number of nodes in mesh. */
#define N 100

/*
 * U_c contains current time layer and U_n
    - next.
 */
double U_c[N], U_n[N];

/* Set initial conditions. */
void initialize(void);
/* Do 1 step of numerical scheme. */
void singleStep(void);
/* Return interpolated value based on
   stencil values. */
```

```
double interpolatedValue(double *u);
/* Fill values in stencil nodes. */
void fillStencilValues(int ind);
/* Print simple representation of current
   U values. */
void debugPrint(void);
/* Save current U values in file for
   Gnuplot processing. */
void debugGnuplot(void);
```

Определение N задаёт количество узлов, на которое разбивается расчётная область $[-1, 1]$, в глобальных массивах U_c и U_n хранятся значения искомой функции на текущем и следующем временном слое соответственно. Функция *initialize*(*void*) производит заполнение массива U_c начальным условием, *singleStep*() выполняет один шаг расчётной схемы, проходя в цикле по всем узлам сетки. Функция *interpolatedValue*(*double* $*$ u) проводит интерполяцию по точкам шаблона и возвращает значение полинома в точке $x_0 = -\lambda\tau$, *debugPrint*() выводит состояние текущего временного слоя в схематичном виде на стандартный поток вывода, *debugGnuplot*() сохраняет в файл для последующего построения графиков в программе Gnuplot. Их реализации приведены в файле **advection.c**.

./src/advection.c

```
#include <stdio.h>
#include <math.h>

#include "advection.h"

#define l 1.0
#define k 0.4
#define h (2.0 / N)
#define tau (k * h / l)
#define M (2.0 / h / k)

/*
 * <stencil> contains numbers of nodes
 * and <stencil_values> - values in this
```

```
     nodes.
  */                                                          15
                                                              16
// 1st order.                                                 17
#ifdef FIRST_ORDER                                            18
#define S 1                                                   19
int stencil[S + 1] = {-1, 0};                                 20
double stencil_values[S + 1];                                 21
#endif                                                        22
// 2nd order.                                                 23
#ifdef SECOND_ORDER                                           24
#define S 2                                                   25
int stencil[S + 1] = {-1, 0, 1};                              26
double stencil_values[S + 1];                                 27
#endif                                                        28
// 3rd order.                                                 29
#ifdef THIRD_ORDER                                            30
#define S 3                                                   31
int stencil[S + 1] = {-2, -1, 0, 1};                          32
double stencil_values[S + 1];                                 33
#endif                                                        34
                                                              35
int main() {                                                  36
  unsigned int step;                                          37
  initialize();                                               38
  debugPrint();                                               39
  for (step = 0; step < M; step++)                            40
    singleStep();                                             41
  debugPrint();                                               42
  debugGnuplot();                                             43
  return 0;                                                   44
}                                                             45
                                                              46
// Rect impulse as initial data.                              47
#ifdef ROUGH_INITIAL                                          48
void initialize(void) {                                       49
  unsigned int ind;                                           50
  for (ind = 0; ind < N; ind++)                               51
```

```
    if ((ind >= N / 4) && (ind <= 3 * N /
        4)) {
      U_c[ind] = 1.0;
      U_n[ind] = 0.0;
    } else {
      U_c[ind] = 0.0;
      U_n[ind] = 0.0;
    }
}
#endif

// sin^4(PI*x), [-1,1] as initial data.
#ifdef SMOOTH_INITIAL
void initialize(void) {
  unsigned int ind;
  double x;
  for (ind = 0; ind < N; ind++) {
    x = -1.0 + ind * h;
    U_c[ind] = pow(sin(M_PI * x), 4.0);
    U_n[ind] = 0.0;
  }
}
#endif

void singleStep(void) {
  int ind;
  for (ind = 0; ind < N; ind++) {
    fillStencilValues(ind);
    U_n[ind] =
        interpolatedValue(stencil_values);
  }
  // FIXME For simplisity - copy U_c = U_n.
  for (ind = 0; ind < N; ind++)
    U_c[ind] = U_n[ind];
}

#ifdef FIRST_ORDER
double interpolatedValue(double *u) {
```

```
    // Linear interpolation: y = a * x + b
    double a = (u[1] - u[0]) / h;
    double b = u[1];
    double x = -l * tau;
    double val = a * x + b;
    return val;
}
#endif

#ifdef SECOND_ORDER
double interpolatedValue(double *u) {
    // Quadratic interpolation: y = a * x^2
        + b * x + c
    double a = (u[2] - 2.0 * u[1] + u[0]) /
        2.0 / h / h;
    double b = (u[2] - u[0]) / 2.0 / h;
    double c = u[1];
    double x = -l * tau;
    double val = a * x * x + b * x + c;
    return val;
}
#endif

#ifdef THIRD_ORDER
double interpolatedValue(double *u) {
    // Qubic interpolation: y = a * x^3 + b
        * x^2 + c * x + d
    double a = (u[3] + 3.0 * u[1] - 3.0 *
        u[2] - u[0]) / 6.0 / h / h / h;
    double b = (u[3] - 2.0 * u[2] + u[1]) /
        2.0 / h / h;
    double c = (2.0 * u[3] + 3.0 * u[2] -
        6.0 * u[1] + u[0]) / 6.0 / h;
    double d = u[2];
    double x = -l * tau;
    double val = a * x * x * x + b * x * x +
        c * x + d;
    return val;
```

```
}
#endif

void fillStencilValues(int ind) {
  unsigned int j;
  int ind_new;
  for (j = 0; j < S + 1; j++) {
    ind_new = ind + stencil[j];
    if (ind_new < 0)
      stencil_values[j] = U_c[ind_new + N];
    else if (ind_new > N - 1)
      stencil_values[j] = U_c[ind_new - N];
    else
      stencil_values[j] = U_c[ind_new];
  }
}

void debugPrint(void) {
  unsigned int ind;
  for (ind = 0; ind < N; ind++)
#define eps 0.1
    if ((U_c[ind] < eps) && (U_c[ind] >
       -eps))
      printf("_");
    else
      printf("|");
  printf("\n");
}

void debugGnuplot(void) {
  FILE *pFile;
  pFile = fopen("plot.dat", "w");
  unsigned int ind;
  for (ind = 0; ind < N; ind ++)
    fprintf(pFile, "%f\t%f\n", -1.0 + h *
       ind, U_c[ind]);
  fclose(pFile);
}
```

Используемые обозначения: $l = \lambda$, h – шаг сетки, tau – шаг по времени, $k = \frac{l*tau}{h}$, M – число шагов по времени. Реализовано два вида начальных условий, используемых в зависимости от флагов компиляции:

1. ROUGH_INITIAL – разрывный прямоугольный единичный импульс, заданный в центре области.

2. SMOOTH_INITIAL – гладкая функция $sin^4(\pi x)$, заданная на всей области.

Для прозрачной реализации процедуры интерполяции по заданному шаблону применён специальный приём. В массив *int stencil*[] в зависимости от флагов компиляции записываются смещения (в индексах) узлов относительно центрального, соответствующие используемому шаблону. Дополнительно выделяется массив *double stencil_values*[], в который перед обработкой каждого узла загружаются значения из массива U_c, соответствующие точкам шаблона. При этом реализуются циклические граничные условия, т.е. справа от крайнего правого узла располагается крайний левый узел.

Для компиляции программы используется скрипт **build.sh**.

./src/build.sh

```
#!/bin/bash                                          1
gcc -g -Wall -DSECOND_ORDER                          2
    -DSMOOTH_INITIAL advection.c
    advection.h -lm -o advection
```

Распределение искомой функции по области расчёта, сохранённое функцией *debugGnuplot*(*void*), может быть визуализировано, например, скриптом **plot.plot**.

./src/plot.plot

```
set terminal png                                     1
set output "plot.png"                                2
set xlabel "Coordinate"                              3
set ylabel "Value"                                   4
plot "plot.dat" using 1:2 title "Solution"           5
```

На рис. 2 приведено сравнение численных решений, полученных схемами на расширенных шаблонах 1-3 порядка. В качестве начального условия использовался разрывный прямоугольный импульс, заданный в центре расчётной области. Представлено распределение искомой функции после одного полного оборота.

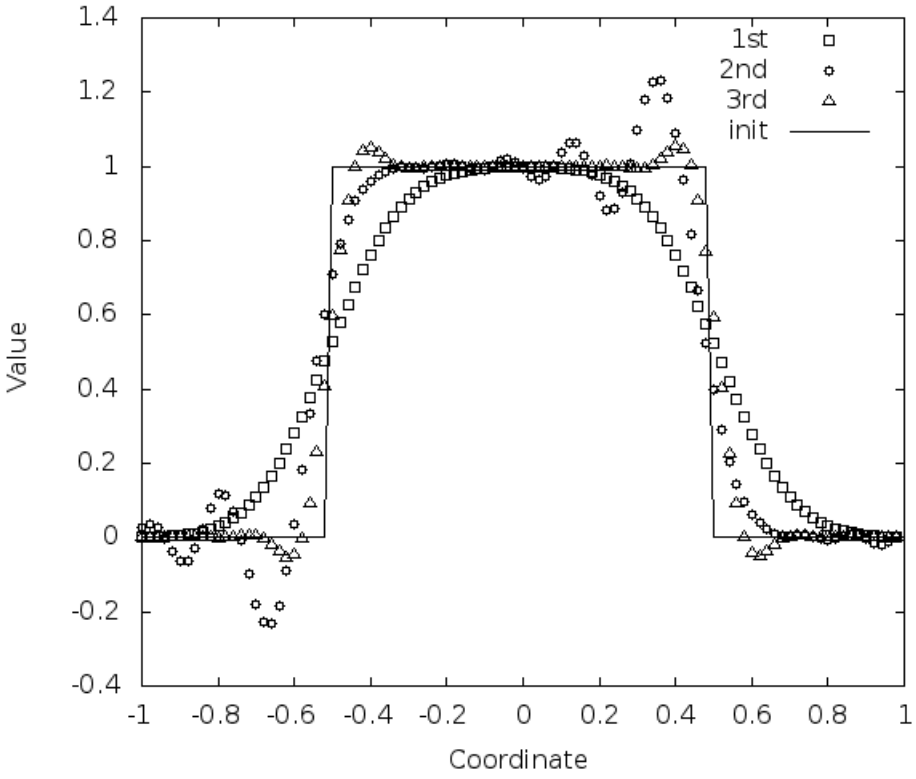

Рис. 2. Сравнение схем 1–3 порядков на расширенных шаблонах

▲ Исследовательские задания

1. Доработайте программу **Advection** для реализации схемы 4-го порядка согласно теории, изложенной в предыдущей главе. Проверьте поведение схемы на гладких и разрывных решениях.

2. Доработайте программу **Advection** для реализации схем повышенного порядка, основанных на расширении сеточного шаблона. Проверьте поведение схем на гладких и разрывных решениях.

1.2. Компактные схемы

1.2.1. Построение и анализ

Альтернативой расширению пространственного шаблона с целью повышения порядка аппроксимации разностной схемы является использование продолженных систем уравнений. При этом используются дифференциальные следствия исходных уравнений. Схемы, построенные с применением данного подхода, называются *компактными* за счёт компактности разностного шаблона.

Введём в рассмотрение дополнительно к уравнению (1) его дифференциальное следствие:

$$\nu_t + \lambda \nu_x = 0, \tag{12}$$

где $\nu(x,t) = u_x(x,t)$ – дополнительная искомая функция. При этом в качестве начального условия может быть записано выражение $\nu(x,0) = u_x(x,0)$.

Рассмотрим двухточечный по пространству шаблон (рис. 1). На нём на текущем временном слое известны как значения функции $u(x,t)$ – u_m^n и u_{m-1}^n, так и функции $\nu(x,t)$ - ν_m^n и ν_{m-1}^n в узлах x_m^n и x_{m-1}^n. Таким образом, по их значениям может быть однозначно восстановлен полином третьего порядка с учётом:

$$\begin{aligned} u^n(x) = F_3(x) = a_3 x^3 + b_3 x^2 + c_3 x + d_3, \\ \nu^n(x) = F_3'(x) = 3a_3 x^2 + 2b_3 x + c_3. \end{aligned} \tag{13}$$

Коэффициенты a_3, b_3, c_3 и d_3 равны соответственно:

$$\begin{aligned} a_3 = \frac{\nu_m^n + \nu_{m-1}^n}{h^2} - 2\frac{u_m^n - u_{m-1}^n}{h^3}, \\ b_3 = \frac{2\nu_m^n + \nu_{m-1}^n}{h} - 3\frac{u_m^n - u_{m-1}^n}{h^2}, \\ c_3 = \nu_m^n, \\ d_3 = u_m^n. \end{aligned}$$

Добавим к рассматриваемому шаблону точку x^n_{m+1}. В таком случае количество известных величин увеличивается до шести, и становится возможным восстановление полинома пятого порядка, который в общем виде представим формулой

$$F_5(x) = a_5x^5 + b_5x^4 + c_5x^3 + d_5x^2 + e_5x + f_5. \qquad (14)$$

Его коэффициенты вычисляются по формулам

$$a_5 = -\frac{3u^n_{m+1} - 3u^n_{m-1} - 4h\nu^n_m - h\nu^n_{m+1} - h\nu^n_{m-1}}{4h^5},$$
$$b_5 = -\frac{-4u^n_m + 2u^n_{m+1} + 2u^n_{m-1} - h\nu^n_{m+1} + h\nu^n_{m-1}}{4h^4},$$
$$c_5 = -\frac{-5u^n_{m+1} + 5u^n_{m-1} + 8h\nu^n_m + h\nu^n_{m+1} + h\nu^n_{m-1}}{4h^3},$$
$$d_5 = -\frac{8u^n_m - 4u^n_{m+1} - 4u^n_{m-1} + h\nu^n_{m+1} - h\nu^n_{m-1}}{4h^2},$$
$$e_5 = \nu^n_m,$$
$$f_5 = u^n_m.$$

1.2.2. Реализация на языке Си

По сравнению с программной реализацией схем на расширенном шаблоне необходимо дополнительно хранить значения производной решения. Для этого вводятся массивы dU_c и dU_n. Кроме того, функция $interpolatedValue$ принимает в аргументах значения производной, и реализована функция $interpolatedDValue(double * u, double * du)$, восстанавливающая значение производной. Прототипы функций приведены в файле **advection_ext.h**, а их реализация – в файле **advection_ext.c**.

./src/advection_ext.h

```
/* Number of nodes in mesh. */                          1
#define N 100                                           2
                                                        3
/*                                                      4
 * U_c contains current time layer and U_n              5
    - next,
 * and dU - derivatives.                                6
```

```
 */
double U_c[N], U_n[N];
double dU_c[N], dU_n[N];

/* Set initial conditions. */
void initialize(void);
/* Do 1 step of numerical scheme. */
void singleStep(void);
/* Return interpolated U value based on
   stencil values. */
double interpolatedValue(double *u, double
   *du);
/* Return interpolated dU value based on
   stencil values. */
double interpolatedDValue(double *u,
   double *du);
/* Fill values in stencil nodes. */
void fillStencilValues(int ind);
/* Print simple representation of current
   U values. */
void debugPrint(void);
/* Print simple representation of current
   U values. */
void debugGnuplot(void);
```

./src/advection_ext.c

```
#include <stdio.h>
#include <math.h>

#include "advection_ext.h"

#define l 1.0
#define k 0.4
#define h (2.0 / N)
#define tau (k * h / l)
#define M (2.0 / h / k)

// 3rd order.
```

```
#ifdef THIRD_ORDER
#define S 1
int stencil[S + 1] = {-1, 0};
double stencil_values[S + 1];
double stencil_Dvalues[S + 1];
#endif

int main() {
  unsigned int step;
  initialize();
  debugPrint();
  for (step = 0; step < M; step++) {
    singleStep();
  }
  debugPrint();
  debugGnuplot();
  return 0;
}

// Rect impulse as initial data.
#ifdef ROUGH_INITIAL
void initialize(void) {
  unsigned int ind;
  for (ind = 0; ind < N; ind++) {
    if ((ind >= N / 4) && (ind <= 3 * N /
        4))
      U_c[ind] = 1.0;
    else
      U_c[ind] = 0.0;
    U_n[ind] = 0.0;
    dU_c[ind] = 0.0;
    dU_n[ind] = 0.0;
  }
}
#endif

// sin^4(PI*x), [-1,1] as initial data.
#ifdef SMOOTH_INITIAL
```

```
void initialize(void) {
  unsigned int ind;
  for (ind = 0; ind < N; ind++) {
    double x = -1.0 + ind * h;
    U_c[ind] = pow(sin(M_PI * x), 4.0);
    U_n[ind] = 0.0;
    dU_c[ind] = 4.0 * M_PI * pow(sin(M_PI
        * x), 3.0) * cos(M_PI * x);
    dU_n[ind] = 0.0;
  }
}
#endif

void singleStep(void) {
  int ind;
  for (ind = 0; ind < N; ind++) {
    fillStencilValues(ind);
    // FIXME Remove calculating a_i twice.
    U_n[ind] =
        interpolatedValue(stencil_values,
        stencil_Dvalues);
    dU_n[ind] =
        interpolatedDValue(stencil_values,
        stencil_Dvalues);
  }
  // FIXME For simplisity copy U_c = U_n,
     dU_c = dU_n.
  for (ind = 0; ind < N; ind++) {
    U_c[ind] = U_n[ind];
    dU_c[ind] = dU_n[ind];
  }
}

#ifdef THIRD_ORDER
double interpolatedValue(double *u, double
   *du) {
  // Cubic interpolation: y = a * x^3 + b
     * x^2 + c * x + d
```

```
  double a = (du[1] + du[0]) / h / h - 2.0 * (u[1] - u[0]) / h / h / h;
  double b = (2.0 * du[1] + du[0]) / h - 3.0 * (u[1] - u[0]) / h / h;
  double c = du[1];
  double d = u[1];
  double x = -l * tau;
  double val = a * x * x * x + b * x * x + c * x + d;
  return val;
}
double interpolatedDValue(double *u, double *du) {
  // Cubic interpolation: dy = 3 * a * x^2 + 2 * b * x + c
  double a = (du[1] + du[0]) / h / h - 2.0 * (u[1] - u[0]) / h / h / h;
  double b = (2.0 * du[1] + du[0]) / h - 3.0 * (u[1] - u[0]) / h / h;
  double c = du[1];
  double x = -l * tau;
  double val = 3.0 * a * x * x + 2.0 * b * x + c;
  return val;
}
#endif

void fillStencilValues(int ind) {
  unsigned int j;
  int ind_new;
  for (j = 0; j < S + 1; j++) {
    ind_new = ind + stencil[j];
    if (ind_new < 0) {
      stencil_values[j] = U_c[ind_new + N];
      stencil_Dvalues[j] = dU_c[ind_new + N];
    } else if (ind_new > N - 1) {
      stencil_values[j] = U_c[ind_new - N];
```

```
        stencil_Dvalues[j] = dU_c[ind_new -
            N];
      } else {
        stencil_values[j] = U_c[ind_new];
        stencil_Dvalues[j] = dU_c[ind_new];
      }
    }
  }

void debugPrint(void) {
  unsigned int ind;
  for (ind = 0; ind < N; ind++)
#define eps 0.1
    if ((U_c[ind] < eps) && (U_c[ind] >
        -eps))
      printf("_");
    else
      printf("|");
  printf("\n");
}

void debugGnuplot(void) {
  FILE *pFile;
  pFile = fopen("plot.dat", "w");
  unsigned int ind;
  for (ind = 0; ind < N; ind ++)
    fprintf(pFile, "%f\t%f\n", -1.0 + h *
        ind, U_c[ind]);
  fclose(pFile);
}
```

Дополнительно к массиву *double stencil_values*[] введён массив *double stencil_Dvalues*[], в который перед обработкой каждого узла загружаются значения из массива dU_c, соответствующие точкам шаблона. Отметим, что в случае разрывного начального условия $ROUGH\_INITIAL$ на нулевом слое может быть использовано нулевое начальное условие на значения производной, в случае $SMOOTH\_INITIAL$ – $\nu(x,0) = u'_x(x,0)$.

На рис. 3 приведено сравнение численного решения, полученного компактной схемой третьего порядка, с аналитическим решением. В качестве начального условия использовался разрывный прямоугольный импульс, заданный в центре расчётной области. Представлено распределение искомой функции после одного полного оборота.

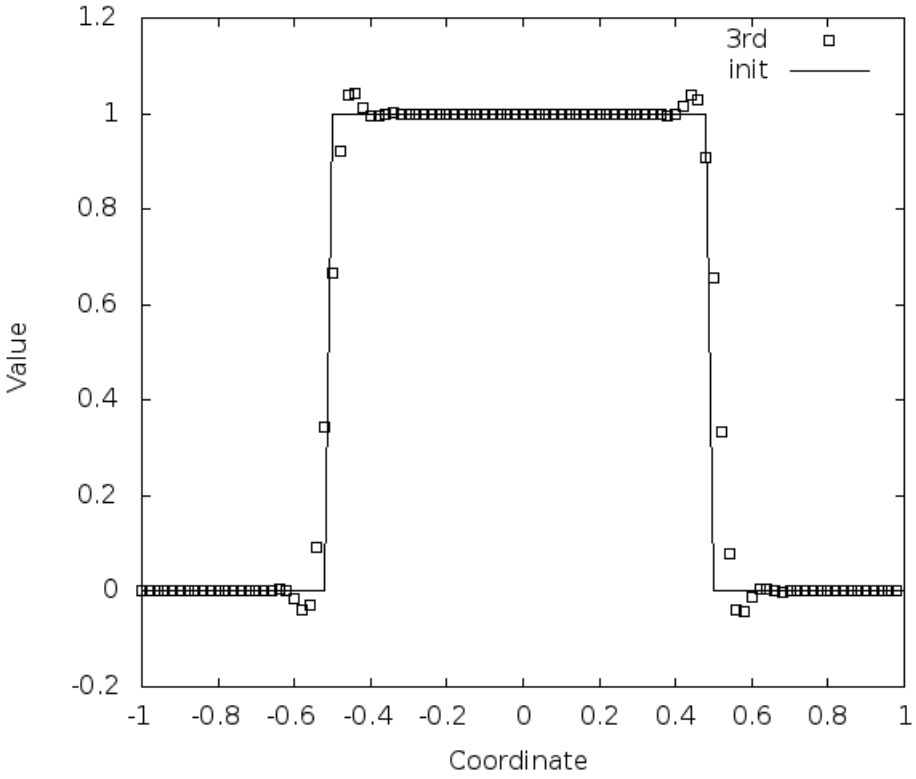

Рис. 3. Сравнение численного решения компактной схемой с аналитическим решением

▲ Исследовательские задания

1. Измените программу **Advection_ext** так, чтобы убрать лишнее повторное вычисление коэффициентов полинома при расчёте значений функции и её производной.

2. Доработайте программу **Advection_ext** для реализации схемы 5-го порядка согласно теории, изложенной в предыдущей главе. Проверьте поведение схемы на гладких и разрывных решениях.

3. Доработайте программу **Advection_ext** для реализации компактных схем повышенного порядка, основанных на расширении сеточного шаблона. Отметим, что некоторые из них могут не обладать свойством устойчивости. Можно ли при данном подходе построить схему чётного порядка сходимости? Какими особенностями она будет обладать? Проверьте поведение схем на гладких и разрывных решениях.

4. Проведите комплексное сравнение времени работы программ **Advection** и **Advection_ext**. Исследуйте вопрос, какую из схем оптимальнее (по времени расчёта) использовать для получения заданной изначально ошибки на аналитическом решении.

1.3. Задачи повышенной сложности

Как было сказано ранее, множество задач из различных областей науки может быть сведено к последовательному решению линейных одномерных уравнений переноса. Очевидно, что реальные практические задачи существенно не одномерны, заданы на сложных пространственных областях и т.д. В настоящей главе покажем подход, который может быть применён для их решения.

Для простоты изложения будем рассматривать задачу двумерного переноса примеси в потоке жидкости. Данный процесс описывается уравнением переноса вида

$$u_t + \lambda_x u_x + \lambda_y u_y = 0. \tag{15}$$

где $u(x, y, t)$ – искомая функция, $\begin{bmatrix} \lambda_x \\ \lambda_y \end{bmatrix}$ – вектор скорости переноса. Для его решения используем подход расщепления по направлениям. При этом решение исходного уравнения (15) заменяется последовательным решением двух уравнений (шаг X и

шаг Y соответственно):

$$\begin{aligned} u_t + \lambda_x u_x = 0, \\ u_t + \lambda_y u_y = 0. \end{aligned} \tag{16}$$

Каждое из уравнений (16) может быть решено любой из схем, построенных на расширенном шаблоне. При этом для шага Y входными данными будет являться распределение, полученное после выполнения шага X, и наоборот. Отметим, что данный подход применим как на квадратных, так и на параллелепипедных расчётных сетках.

▲ **Задача 1.** Реализуйте программу **Advection_2D**, в которой бы моделировался процесс двумерного переноса схемами на расширенном шаблоне. Проведите серию тестовых расчётов как на гладком, так и на разрывном решении. Какие особенности численного решения вы наблюдаете?

▲ **Задача 2.** По аналогии с двумерным случаем реализуйте решение трёхмерного уравнения линейного переноса с помощью подхода расщепления по направлениям в виде программы **Advection_3D**.

Основным отличием способа построения одномерных компактных схем от схем на расширенном шаблоне является дополнительное хранение и пересчёт значений производной решения. Попробуем использовать напрямую подход расщепления по направлениям. На шаге X имеем одномерное уравнение переноса, которое, следуя методике построения продолженных схем, дополним его дифференциальным следствием (производная по X):

$$\begin{cases} u_t + \lambda_x u_x = 0, \\ (u_x)_t + \lambda_x (u_x)_x = 0. \end{cases} \tag{17}$$

Данная система может быть решена компактной схемой 3-го или 5-го (возможно выше) порядка точности, после чего будут известны значения $u^{n+1/2}$ и $u_x^{n+1/2}$ на промежуточном временном слое. На шаге по Y аналогично записываем решаемую систему уравнений в виде

$$\begin{cases} u_t + \lambda_y u_y = 0, \\ (u_y)_t + \lambda_y (u_y)_y = 0. \end{cases} \tag{18}$$

Но для её решения необходимо знание значений $u_y^{n+1/2}$! При этом система (17) может быть дополнена следствием:

$$(u_y)_t + \lambda_x (u_y)_x = 0. \tag{19}$$

Для нахождения $u_y^{n+1/2}$ могут быть использованы два принципиально разных подхода:

1. Решение уравнения переноса (19) схемой первого порядка.
2. Дополнение системы (17) ещё одним дифференциальным следствием $(u_{xy})_t + \lambda_x (u_{xy})_x = 0$ и решение двух независимых продолженных систем уравнений любой компактной схемой.

▲ **Задача 3.** Реализуйте программу **Advection_ext_2D**, в которой бы моделировался процесс двумерного переноса компактными схемами. Рассмотрите два случая:

1. Решение одной продолженной системы и использование схемы первого порядка.
2. Решение двух продолженных систем.

Проведите серию тестовых расчётов как на гладком, так и на разрывном решении. Оцените порядок сходимости построенных схем. Проведите их сравнительный анализ, перечислите их плюсы и минусы.

▲ **Задача 4.** Постройте компактные схемы для решения трёхмерного линейного уравнения переноса. Какие варианты их построения возможны? Какие их плюсы и минусы? Попробуйте оценить порядок сходимости на гладком решении. Что является лимитирующим фактором?

2. Система уравнений акустики

2.1. Переход в инварианты Римана

Рассмотрим задачу о распространении волн давления малой амплитуды в газе в одномерной постановке. Будем считать, что равновесное состояние характеризуется нулевой скоростью течения газа, плотностью ρ_0 и давлением $P(\rho_0)$. Неизвестными функциями являются давление $p(x,t)$ и скорость $u(x,t)$, как малые величины относительно равновесных значений. Если ввести обозначение $K_0 = \rho_0 P'(\rho_0)$, то динамические процессы, происходящие в среде, могут быть описаны линейной системой дифференциальных уравнений:

$$\begin{bmatrix} p \\ u \end{bmatrix}_t + \begin{bmatrix} 0 & K_0 \\ \frac{1}{\rho_0} & 0 \end{bmatrix} \begin{bmatrix} p \\ u \end{bmatrix}_x = 0. \tag{20}$$

Собственные значения матрицы данной системы $\lambda_{1,2} = \pm c_0$, где $c_0 = \sqrt{P'(\rho_0)}$. Матрица S, составленная из собственных векторов, и обратная к ней S^{-1} имеют вид:

$$S = \begin{bmatrix} -Z_0 & Z_0 \\ 1 & 1 \end{bmatrix}, S^{-1} = \frac{1}{2Z_0} \begin{bmatrix} -1 & Z_0 \\ 1 & Z_0 \end{bmatrix}, \tag{21}$$

где $Z_0 = \rho_0 c_0$.

Таким образом, система (20) может быть переписана в виде

$$\begin{bmatrix} p \\ u \end{bmatrix}_t + S \begin{bmatrix} -c_0 & 0 \\ 0 & c_0 \end{bmatrix} S^{-1} \begin{bmatrix} p \\ u \end{bmatrix}_x = 0. \tag{22}$$

Домножая обе части (22) на S^{-1} слева и вводя замену переменных

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = S^{-1} \begin{bmatrix} p \\ u \end{bmatrix} = \frac{1}{2Z_0} \begin{bmatrix} -p + Z_0 u \\ p + Z_0 u \end{bmatrix}, \tag{23}$$

получим следующую систему уравнений:

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_t + \begin{bmatrix} -c_0 & 0 \\ 0 & c_0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_x = 0. \tag{24}$$

Таким образом, для решения задачи о распространении волн малой амплитуды в газе достаточно решить два независимых уравнения переноса в переменных w_1, w_2 и перейти обратно к переменным p и u.

2.2. Построение компактных схем

Рассмотрим систему уравнений переноса (24). Введём дополнительные искомые функции с соответствующими граничными условиями:

$$\begin{cases} \nu_1(x,t) = \frac{\partial w_1}{\partial x}(x,t), \\ \nu_2(x,t) = \frac{\partial w_2}{\partial x}(x,t), \\ \nu_1(x,0) = \frac{\partial w_1}{\partial x}(x,0), \\ \nu_2(x,0) = \frac{\partial w_2}{\partial x}(x,0). \end{cases} \tag{25}$$

Системы (24) и (25) вместе образуют две независимые продолженные системы для уравнений переноса w_1 и w_2, которые можно решать, используя полученные ранее компактные схемы. Следуя (3), можно записать, что

$$w_1(x_m, t_n + \tau) = w_1(x_m + c_0\tau, t_n),$$
$$w_2(x_m, t_n + \tau) = w_2(x_m - c_0\tau, t_n).$$

Таким образом, минимальный шаблон для решения акустической системы включает в себя четыре точки $x_m^{n+1}, x_m^n, x_{m-1}^n, x_{m+1}^n$ [10]. Зная матрицы перехода (27), можно рассчитать значения инвариантов Римана на текущем слое по времени через переменные задачи:

$$w_{1,m}^n = \frac{1}{2Z_0}(-p_m^n + Z_0 u_m^n),$$
$$w_{2,m}^n = \frac{1}{2Z_0}(p_m^n + Z_0 u_m^n).$$

Для решения каждой из продолженных систем можно воспользоваться компактной схемой третьего порядка точности. То-

гда для w_2 будем иметь

$$
\begin{aligned}
w_2^n(x) &= a_2x^3 + b_2x^2 + c_2x + d_2,\\
\nu_2^n(x) &= 3a_2x^2 + 2b_2x + c_2,\\
a_2 &= \frac{\nu_{2,m}^n + \nu_{2,m-1}^n}{h^2} - 2\frac{w_{2,m}^n - w_{2,m-1}^n}{h^3},\\
b_2 &= \frac{2\nu_{2,m}^n + \nu_{2,m-1}^n}{h} - 3\frac{w_{2,m}^n - w_{2,m-1}^n}{h^2},\\
c_2 &= \nu_{2,m}^n,\\
d_2 &= w_{2,m}^n,\\
w_{2,m}^{n+1} &= w_2^n(x_m - c_0\tau),\\
\nu_{2,m}^{n+1} &= \nu_2^n(x_m - c_0\tau).
\end{aligned}
$$

А для w_1 будем иметь (*соответствующие выкладки проделайте самостоятельно*):

$$
\begin{aligned}
w_1^n(x) &= a_1x^3 + b_1x^2 + c_1x + d_1,\\
\nu_1^n(x) &= 3a_1^x2 + 2b_1x + c_1,\\
a_1 &= \frac{\nu_{1,m+1}^n + \nu_{1,m}^n}{h^2} - 2\frac{w_{1,m+1}^n - w_{1,m}^n}{h^3},\\
b_1 &= -\frac{\nu_{1,m+1}^n + 2\nu_{1,m}^n}{h} + 3\frac{w_{1,m+1}^n - w_{1,m}^n}{h^2},\\
c_1 &= \nu_{1,m}^n,\\
d_1 &= w_{1,m}^n,\\
w_{1,m}^{n+1} &= w_1^n(x_m + c_0\tau),\\
\nu_{1,m}^{n+1} &= \nu_1^n(x_m + c_0\tau).
\end{aligned}
$$

Зная матрицы перехода (27), можно рассчитать значения переменных задачи через значения инвариантов Римана на следующем слое по времени:

$$
\begin{aligned}
p_m^{n+1} &= Z_0(-w_{1,m}^{n+1} + w_{2,m}^{n+1}),\\
u_m^{n+1} &= w_{1,m}^{n+1} + w_{2,m}^{n+1}.
\end{aligned}
$$

▲ Исследовательские задания

1. Постройте компактную схему пятого порядка точности для системы уравнений акустики. Сколько точек в этом случае

входит в её шаблон? Предложите варианты асимметричных компактных схем, построенных на меньшем числе точек. В чём, по-вашему, заключается их недостаток?

2. Используя подход расширения пространственного шаблона для повышения точности схемы, постройте схемы с 1 до 5 порядка включительно.

2.3. Реализация на языке C

Задача рассматривалась изначально в инвариантах Римана, т.е. переход от скорости и давления к ним, а также обратный переход не реализовывались. Ввиду того, что знаки коэффициентов перед производной по координате в двух уравнениях переноса на инварианты различны, были соответствующим образом расширены функции *interpolatedValue*, *interpolatedDValue*, *fillStencilValues* и *singleStep*. Прототипы функций приведены в файле **acoustic_ext.h**, а их реализация – в файле **acoustic_ext.c**. Для сборки программы может быть использован скрипт **build_ac_ext.sh**.

./src/build_ac_ext.sh

```
#!/bin/bash
gcc -g -Wall acoustic_ext.c acoustic_ext.h
    -lm -o acoustic_ext
```

./src/acoustic_ext.h

```
/* Number of nodes in mesh. */
#define N 100

/*
 * *_c contains current time layer and *_n
    - next,
 * and d* - derivatives.
 * Currently solve in Riemann invariants.
 */
double W1_c[N], W2_c[N], W1_n[N], W2_n[N];
double dW1_c[N], dW2_c[N], dW1_n[N],
    dW2_n[N];
```

```
                                                    11
/* Set initial conditions. */                       12
void initialize(void);                              13
/* Do 1 step of numerical scheme. */                14
void singleStep(void);                              15
/* Return interpolated u value based on             16
   stencil values. */
double interpolatedValue(double *u, double          17
   *du, int dir);
/* Return interpolated du value based on            18
   stencil values. */
double interpolatedDValue(double *u,                19
   double *du, int dir);
/* Fill values in stencil nodes. */                 20
void fillStencilValues(int ind);                    21
/* Print simple representation of current           22
   * values. */
void debugPrint(void);                              23
```

./src/acoustic_ext.c

```
#include <stdio.h>                                  1
#include <math.h>                                   2
                                                    3
#include "acoustic_ext.h"                           4
                                                    5
#define c_0 1                                       6
#define Z_0 1 // Z_0=rho_0*c_0                      7
#define k 0.4                                       8
#define h (2.0 / N)                                 9
#define tau (k * h / c_0)                           10
#define M (0.25 / h / k)                            11
                                                    12
// 3rd order extended scheme is realized.           13
#define S 1                                         14
int stencil_l[S + 1] = {-1, 0};                     15
int stencil_r[S + 1] = {0, 1};                      16
double stencil_values_w1[S + 1];                    17
double stencil_values_w2[S + 1];                    18
```

```
double stencil_Dvalues_w1[S + 1];                    19
double stencil_Dvalues_w2[S + 1];                    20
#define LEFT 0                                       21
#define RIGHT 1                                      22
                                                     23
int main() {                                         24
  unsigned int step;                                 25
  initialize();                                      26
  debugPrint();                                      27
  for (step = 0; step < M; step++) {                 28
    singleStep();                                    29
  }                                                  30
  debugPrint();                                      31
  return 0;                                          32
}                                                    33
                                                     34
// Rect impulse as initial data.                     35
void initialize(void) {                              36
  unsigned int ind;                                  37
  for (ind = 0; ind < N; ind++) {                    38
    if ((ind >= N / 4) && (ind <= 3 * N /            39
        4)) {
      W1_c[ind] = 1.0 / (2.0 * Z_0);                 40
      W2_c[ind] = -1.0 / (2.0 * Z_0);                41
    } else {                                         42
      W1_c[ind] = 0.0;                               43
      W2_c[ind] = 0.0;                               44
    }                                                45
    W1_n[ind] = 0.0;                                 46
    W2_n[ind] = 0.0;                                 47
    dW1_c[ind] = 0.0;                                48
    dW2_c[ind] = 0.0;                                49
    dW1_n[ind] = 0.0;                                50
    dW2_n[ind] = 0.0;                                51
  }                                                  52
}                                                    53
                                                     54
void singleStep(void) {                              55
```

```
  int ind;                                                    56
  for (ind = 0; ind < N; ind++) {                             57
    fillStencilValues(ind);                                   58
    // FIXME Remove calculating a_i twice.                    59
    W1_n[ind] =                                               60
        interpolatedValue(stencil_values_w1,
        stencil_Dvalues_w1, RIGHT);
    dW1_n[ind] =                                              61
        interpolatedDValue(stencil_values_w1,
        stencil_Dvalues_w1, RIGHT);
    W2_n[ind] =                                               62
        interpolatedValue(stencil_values_w2,
        stencil_Dvalues_w2, LEFT);
    dW2_n[ind] =                                              63
        interpolatedDValue(stencil_values_w2,
        stencil_Dvalues_w2, LEFT);
  }                                                           64
  // FIXME For simplisity copy Wi_c =                         65
     Wi_n, dWi_c = dWi_n.
  for (ind = 0; ind < N; ind++) {                             66
    W1_c[ind] = W1_n[ind];                                    67
    dW1_c[ind] = dW1_n[ind];                                  68
    W2_c[ind] = W2_n[ind];                                    69
    dW2_c[ind] = dW2_n[ind];                                  70
  }                                                           71
}                                                             72
                                                              73
double interpolatedValue(double *u, double                    74
   *du, int dir) {
  // Cubic interpolation: y = a * x^3 + b                     75
     * x^2 + c * x + d
  double a, b, c, d, x;                                       76
  if (dir == LEFT) {                                          77
    a =  (du[1] + du[0]) / h / h - 2.0 *                      78
        (u[1] - u[0]) / h / h / h;
    b = (2.0 * du[1] + du[0]) / h - 3.0 *                     79
        (u[1] - u[0]) / h / h;
    c = du[1];                                                80
```

```
    d = u[1];                                          81
    x = -c_0 * tau;                                    82
  } else {                                             83
    a =  (du[1] + du[0]) / h / h - 2.0 *               84
        (u[1] - u[0]) / h / h / h;
    b = -(2.0 * du[0] + du[1]) / h + 3.0 *             85
        (u[1] - u[0]) / h / h;
    c = du[0];                                         86
    d = u[0];                                          87
    x = c_0 * tau;                                     88
  }                                                    89
  double val = a * x * x * x + b * x * x +             90
      c * x + d;
  return val;                                          91
}                                                      92
double interpolatedDValue(double *u,                  93
    double *du, int dir) {
  // Cubic interpolation: dy = 3 * a * x^2             94
      + 2 * b * x + c
  double a, b, c, x;                                   95
  if (dir == LEFT) {                                   96
    a = (du[1] + du[0]) / h / h - 2.0 *                97
        (u[1] - u[0]) / h / h / h;
    b = (2.0 * du[1] + du[0]) / h - 3.0 *              98
        (u[1] - u[0]) / h / h;
    c = du[1];                                         99
    x = -c_0 * tau;                                    100
  } else {                                             101
    a = (du[1] + du[0]) / h / h - 2.0 *                102
        (u[1] - u[0]) / h / h / h;
    b = -(2.0 * du[0] + du[1]) / h + 3.0 *             103
        (u[1] - u[0]) / h / h;
    c = du[0];                                         104
    x = c_0 * tau;                                     105
  }                                                    106
  double val = 3.0 * a * x * x + 2.0 * b *             107
      x + c;
  return val;                                          108
```

```
}                                                                    109
                                                                     110
void fillStencilValues(int ind) {                                    111
  unsigned int j;                                                    112
  int ind_new_l, ind_new_r;                                          113
  for (j = 0; j < S + 1; j++) {                                      114
    ind_new_l = ind + stencil_l[j];                                  115
    if (ind_new_l < 0) {                                             116
      stencil_values_w2[j] =                                         117
          W2_c[ind_new_l + N];
      stencil_Dvalues_w2[j] =                                        118
          dW2_c[ind_new_l + N];
    } else {                                                         119
      stencil_values_w2[j] =                                         120
          W2_c[ind_new_l];
      stencil_Dvalues_w2[j] =                                        121
          dW2_c[ind_new_l];
    }                                                                122
    ind_new_r = ind + stencil_r[j];                                  123
    if (ind_new_r > N - 1) {                                         124
      stencil_values_w1[j] =                                         125
          W1_c[ind_new_r - N];
      stencil_Dvalues_w1[j] =                                        126
          dW1_c[ind_new_r - N];
    } else {                                                         127
      stencil_values_w1[j] =                                         128
          W1_c[ind_new_r];
      stencil_Dvalues_w1[j] =                                        129
          dW1_c[ind_new_r];
    }                                                                130
  }                                                                  131
}                                                                    132
                                                                     133
void debugPrint(void) {                                              134
  unsigned int ind;                                                  135
#define eps 0.1                                                      136
  printf("W1:\n");                                                   137
  for (ind = 0; ind < N; ind++)                                      138
```

```
    if ((W1_c[ind] < eps) && (W1_c[ind] > -eps))
      printf("_");
    else
      printf("|");
  printf("\n");
  printf("W2:\n");
  for (ind = 0; ind < N; ind++)
    if ((W2_c[ind] < eps) && (W2_c[ind] > -eps))
      printf("_");
    else
      printf("|");
  printf("\n");
}
```

Для случая разрывного начального импульса на первом слое использовано нулевое начальное условие на значения производных, необходимых для построения компактной схемы.

▲ Исследовательские задания

1. Модифицируйте программу **Acoustic_ext**, добавив этапы перехода от переменных (p, u) к переменным (w_1, w_2) и обратно.

2. Модифицируйте программу **Acoustic_ext** так, чтобы убрать лишнее повторное вычисление коэффициентов полинома при расчёте значений функции и её производной.

3. Модифицируйте программу **Acoustic_ext** так, чтобы убрать необходимость в различном вычислении коэффициентов полинома для w_1 и w_2. *Подсказка*: отразите симметрично шаблон для w_1.

4. Доработайте программу **Acoustic_ext** для реализации схемы 5-го порядка. Проверьте поведение схемы на гладких и разрывных решениях.

5. *Разработайте программы для решения системы уравнений акустики в двумерном и трёхмерном случаях. *Подсказка*: используйте подход расщепления по направлениям.

3. Система уравнений упругости

3.1. Переход в инварианты Римана

Рассмотрим задачу о распространении волн давления малой амплитуды в упругой среде в одномерной постановке. Аналогично случаю с распространением возмущений в газе неизвестные функции будут отсчитываться от равновесных значений. К ним относится напряжение $\sigma_{xx}(x,t)$ и скорость среды $u(x,t)$. Дополнительно введём обозначения: λ и μ – упругие параметры Лямэ, ρ – плотность среды. Динамические процессы, происходящие в среде, могут быть описаны линейной системой дифференциальных уравнений:

$$\begin{bmatrix} \sigma_{xx} \\ u \end{bmatrix}_t + \begin{bmatrix} 0 & -(\lambda+2\mu) \\ -\frac{1}{\rho} & 0 \end{bmatrix} \begin{bmatrix} \sigma_{xx} \\ u \end{bmatrix}_x = 0. \tag{26}$$

Собственные значения матрицы данной системы $\lambda_{1,2} = \pm c_p$, где $c_p = \sqrt{\frac{\lambda+2\mu}{\rho}}$. Матрица S, составленная из собственных векторов, и обратная к ней S^{-1} имеют вид:

$$S = \begin{bmatrix} Z & -Z \\ 1 & 1 \end{bmatrix}, S^{-1} = \frac{1}{2Z} \begin{bmatrix} 1 & Z \\ -1 & Z \end{bmatrix}, \tag{27}$$

где $Z = \rho c_p$.

Аналогично (23) введением замены переменных

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = S^{-1} \begin{bmatrix} \sigma_{xx} \\ u \end{bmatrix} = \frac{1}{2Z} \begin{bmatrix} \sigma_{xx} + Zu \\ -\sigma_{xx} + Zu \end{bmatrix} \tag{28}$$

исходную систему уравнений (26) можно преобразовать к виду

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_t + \begin{bmatrix} -c_p & 0 \\ 0 & c_p \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_x = 0. \tag{29}$$

Таким образом, для решения задачи о распространении упругих волн малой амплитуды в среде достаточно решить два независимых уравнения переноса в переменных w_1, w_2 и перейти обратно к переменным σ_{xx} и u.

3.2. Исследовательские задания

1. Разработайте программу, выполняющую численное решение системы уравнений упругости в одномерном случае с использованием сеточно-характеристических схем на расширенном шаблоне. Сравните результаты расчётов по методам с различным порядком аппроксимации между собой и с известными аналитическими решениями.

2. Разработайте программу, выполняющую численное решение системы уравнений упругости в одномерном случае с использованием компактных сеточно-характеристических схем. Сравните результаты расчётов по методам с различным порядком аппроксимации между собой и с известными аналитическими решениями.

3. Усовершенствуйте программы из двух предыдущих заданий на двумерный и трёхмерный случаи. Проведите сравнение результатов численных расчётов, полученных различными схемами, между собой и с известными аналитическими решениями для плоских продольных и поперечных волн и для точечного источника. Сопоставьте скорость работы программ, а также объёмы потребляемой ими оперативной памяти. *Подсказка*: используйте подход расщепления по направлениям.

Литература

1. *Холодов А.С.* Численные методы решения уравнений и систем гиперболического типа // Энциклопедия низкотемпературной плазмы // T.VII-1. Ч.2. М.: Янус-К, 2008. С. 141–174.

2. *Петров И.Б., Холодов А.С.* Численное исследование некоторых динамических задач механики деформируемого твёрдого тела сеточно-характеристическим методом // Журнал вычислительной математики и математической физики. 1984. Т. 24, № 5. С. 722–739.

3. *Голубев В.И., Петров И.Б., Хохлов Н.И.* Численное моделирование сейсмической активности сеточно-характеристическим методом // Журнал вычислительной математики и математической физики. 2013. Т. 53, № 10. С. 1709–1720.

4. *Петров И.Б., Хохлов Н.И.* Моделирование задач 3D сейсмики на высокопроизводительных вычислительных системах // Математическое моделирование. 2014. Т. 26, № 1. С. 83–95.

5. *Rusanov V.V.* Calculation of intersection of non-steady shock waves with obstacles // J. Comput. Math. Phys. USSR. 1961. Vol. 1. P. 267–279.

6. *Courant Richard and Lax Peter.* On nonlinear partial differential equations with two independent variables // Communications on Pure and Applied Mathematics. 1949. Vol. 2. P. 255–273.

7. *Магомедов К.М., Холодов А.С.* Сеточно-характеристические численные методы. М.: Наука, 1988. 287 с.

8. *Петров И.Б.* Лекции по вычислительной математике: Учебное пособие / И.Б. Петров, А.И. Лобанов. М.: Интернет-Университет Информационных технологий; БИНОМ. Лаборатория знаний, 2006. 523 с.

9. *Lax P.D., Wendroff B.* System of conservation laws // Comm. Pure and Appl. Math. 1960. Vol. 13, N. 2. P. 217–237.

10. *Shiraishi S., Matsuoka T.* Wave propagation simulation using the CIP method of characteristic equations // Communications in Computational Physics. 2008. Vol. 3, N. 1. P. 121–135.

Учебное издание

СЕТОЧНО-ХАРАКТЕРИСТИЧЕСКИЕ МЕТОДЫ ДЛЯ ЧИСЛЕННОГО РЕШЕНИЯ ЛИНЕЙНЫХ ОДНОМЕРНЫХ ГИПЕРБОЛИЧЕСКИХ УРАВНЕНИЙ И СИСТЕМ

Методические указания

по курсам *Информатика* и *Вычислительная математика*

Составитель **Голубев** Василий Иванович

Редактор *О.П. Котова*. Корректор *Л.В. Себова*
Компьютерная верстка *Н.Е. Кобзева*

Подписано в печать 24.01.2013. Формат 60×84 $^{1}/_{16}$.
Усл. печ. л. 3,25. Уч.-изд.л. 3,0. Тираж 100 экз. Заказ №17.

Федеральное государственное автономное образовательное
учреждение высшего образования «Московский
физико-технический институт(государственный университет)»
141700, Московская обл., г. Долгопрудный, Институтский пер., 9
E-mail: rio@mail.mipt.ru

Отдел оперативной полиграфии «Физтех-полиграф»
141700, Московская обл., г. Долгопрудный, Институтский пер., 9
E-mail: polygraph@mipt.ru